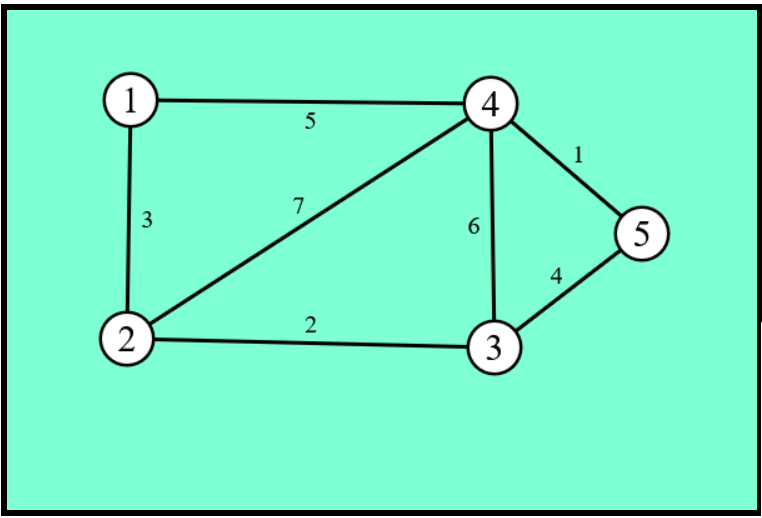
**Алгоритм Краскала**

Алгоритм Краскала решает следующую задачу:

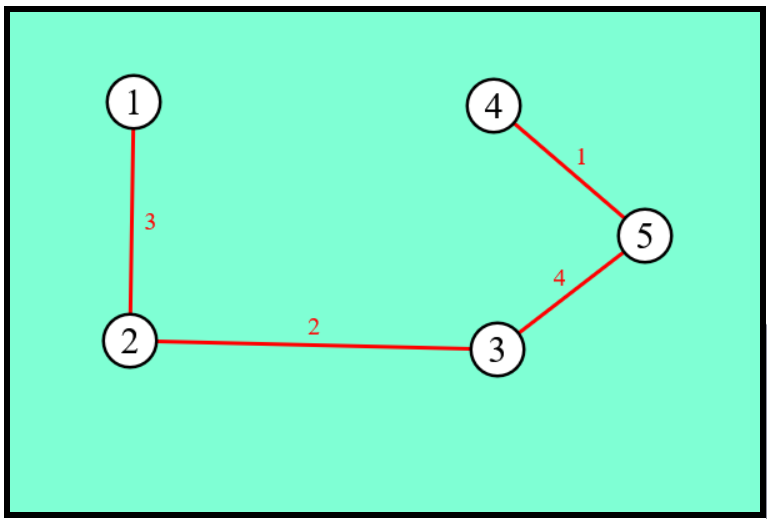
Дан неориентированный взвешенный граф. Он связный, т.е. из любой вершины есть путь в любую. Найти его минимальное остовное дерево – дерево, которое получается из графа удалением некоторых рёбер и имеет минимальную сумму весов рёбер.

Дерево – это связный граф без циклов.

Другими словами, из графа нужно удалить некоторые рёбра так, чтобы граф остался связным, в нём не было циклов и сумма весов оставшихся рёбер была минимальной.



Для данного примера минимальное остовное дерево будет иметь следующий вид (сумма весов 10):



**Алгоритм Краскала**:

Будем перебирать рёбра по возрастанию весов. Если ребро можно добавить в ответ без образования цикла, то добавляем его. Иначе пропускаем.

**Доказательство минимальности построенного дерева:**

Пусть есть отсортированный порядок рёбер по возрастанию весов (если есть равные веса, то такие рёбра упорядочим между собой любым способом). Назовём этот порядок ord.

Алгоритм берёт ребро в дерево тогда и только тогда, когда между его концами нет пути из рёбер, находящихся раньше данного ребра в ord.

Пусть теперь есть остовное дерево с меньшей суммой весов, чем построенное алгоритмом Краскала. Найдём в ord первое ребро, которое добавлено в дерево по алгоритму, но не добавлено в настоящем минимальном дереве (пусть это ребро v-u). Если добавить это ребро в минимальное дерево, образуется цикл. Найдём на этом цикле ребро, которое встречается позже всего в ord. Оно будет встречаться позже ребра v-u, потому что ребро v-u добавлено алгоритмом, а это означает, что между v и u нет пути из рёбер, встречающихся раньше данного.

Тогда можно заменить в минимальном дереве это максимальное ребро на ребро v-u и сумма весов не увеличится.

Таким образом, в любом минимальном остовном дереве, не построенном по алгоритму, можно заменить ребро так, чтобы полученное дерево осталось минимальным, но сумма индексов рёбер в ord уменьшилась. Тогда возьмём минимальное остовное дерево с минимальной суммой этих индексов. В нём нельзя сделать такую замену ребра => оно совпадает с построенным по алгоритму.

**Как проверить, что добавляемое ребро не образует цикл**

Чтобы при добавлении ребра u-v образовался цикл, в графе уже должен быть путь между вершинами u и v. Его можно найти обходом в глубину или в ширину, но получится время работы **O(m2).** Или можно использовать структуру данных «система непересекающихся множеств» (disjoint set union, DSU).

Эта структура решает следующую задачу:

Есть n множеств из одного элемента. Нужно выполнять следующие операции:

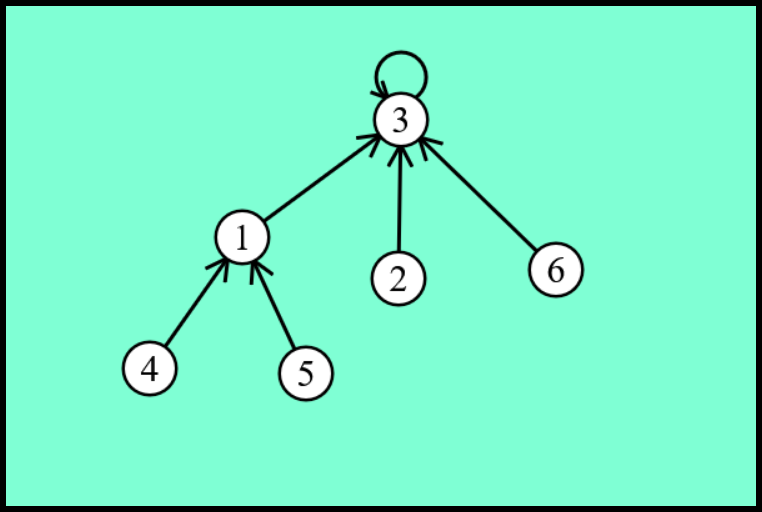
1. Объединить 2 множества, содержащие заданные элементы

2. Проверить, лежат ли 2 элемента в одном множестве

Для алгоритма Краскала этими множествами будут компоненты связности графа. Изначально граф содержит n компонент из одной вершины. При добавлении ребра нужно объединять 2 компоненты. Для проверки того, что добавление ребра создаст цикл, нужно проверить, что 2 вершины лежат в одной компоненте связности.

**Система непересекающихся множеств**

Структура строится следующим образом. Каждое множество будет представлено в виде дерева, в котором для каждой вершины хранится её родитель. Пусть p[i] – это родитель вершины i. Для корня дерева значение p будет самим корнем.



Тогда чтобы объединить 2 множества, нужно подвесить корень одного из них к любой вершине другого (но для уменьшения высоты дерева будем подвешивать к корню). А чтобы проверить, лежат ли 2 вершины в одном множестве, нужно проверить, что у них одинаковый корень.

В этом алгоритме худший случай – когда высота деревьев становится слишком большой, тогда сложно будет искать корень дерева, содержащего данную вершину.

Высота дерева может доходить до **O(n),** если большое дерево всё время подвешивается к дереву из одной вершины. Чтобы уменьшить высоту дерева, применяются 2 оптимизации:

1. Сжатие путей. Когда по вершине нужно найти корень её дерева (пройдя по родителям), можно всех этих родителей переподвешивать к самому корню.

2. Small-to-large. Для каждого корня будем хранить размер его дерева. При объединении двух деревьев будем подвешивать корень меньшего из них к корню большего.

Каждая из этих оптимизаций по отдельности позволяет добиться времени **O(n log n),** а вместе они работают за **O(n α(n)),** где α(n) – обратная функция Аккермана, которая не превосходит 4 при всех разумных значениях n.

С системой непересекающихся множеств алгоритм Краскала работает за **O(m log m)** из-за сортировки. После сортировки сам алгоритм работает за **O(m α(m)).**